

Modeling and visualizing networked multi-core embedded software energy consumption

Steve Kerrison and Kerstin Eder
University of Bristol, United Kingdom
firstname.lastname@bristol.ac.uk

Technical Report, August 2015

Abstract

In this report we present a network-level multi-core energy model and a software development process workflow that allows software developers to estimate the energy consumption of multi-core embedded programs. This work focuses on a high performance, cache-less and timing predictable embedded processor architecture, XS1. Prior modelling work is improved to increase accuracy, then extended to be parametric with respect to voltage and frequency scaling (VFS) and then integrated into a larger scale model of a network of interconnected cores. The modelling is supported by enhancements to an open source instruction set simulator to provide the first network timing aware simulations of the target architecture. Simulation based modelling techniques are combined with methods of results presentation to demonstrate how such work can be integrated into a software developer's workflow, enabling the developer to make informed, energy aware coding decisions. A set of single-, multi-threaded and multi-core benchmarks are used to exercise and evaluate the models and provide use case examples for how results can be presented and interpreted. The models all yield accuracy within an average $\pm 5\%$ error margin.

1 Introduction

An increasing number of embedded systems now express their workloads through concurrent software. The parallelism present in modern devices, in forms such as multi-threading and multiple cores, allow this concurrency to be exploited. This progression towards parallel systems has two main motivations. The first is in response to hitting operating frequency limits, where more work must now be done per clock in order to achieve performance gains in each new device generation. The other uses parallelism to allow work to be completed on time at a lower operating frequency, which can yield significant energy reductions.

However, parallel systems and concurrent software introduce complexities over traditional sequential variants that simply valued "straight-line speed". In particular, synchronisation of and exchanging information between concurrent components can negatively impact parallel performance if done inefficiently as per the well known *Amdahl's Law*. A good understanding of the software's behaviour, coupled with appropriate underlying hardware can overcome this if used correctly.

In embedded systems software, predictability is essential, both in terms of execution time, where real-time deadlines must be met, and in terms of energy consumption, where the supply of energy may be scarce. Time and energy are related through power, and while significant effort is put into timing predictable software, there remains both a lack of intuition and a lack of tools to help software developers determine the energy consumption of their modern embedded software components.

This report presents an energy model for a family of cache-less, time-deterministic, hardware multi-threaded embedded processors, the XMOS XS1-L series, which implements the XS1 architecture. These processors are programmed in a C-like language with message passing present in both the architecture and the programming model. The processors can be assembled into networks of interconnected cores, where the communication paradigm then extends across this network. The energy model must therefore be able to account for software energy consumption within each core as well as the timing and power effects of network traffic. To achieve this and also give developers better energy estimation tools, the following contributions are made:

- A multi-threaded energy model for the XS1-L [15] is extended to include more accurate instruction energy data, through greater instruction profiling and regression tree techniques.
- Support for Voltage and Frequency Scaling (VFS) is integrated into the model, the provide a richer environment for design space exploration by software developers.
- Several new features are added to **axe**, an Instruction Set Simulator (ISS) for the XS1-L, improving its core timing accuracy and introducing network timing behaviour, which has until now not been present in any simulators for these devices.
- The energy consumption of network communication is profiled, in order to extend the energy model to account for communication between multi-cores.

These contributions allow traces from the **axe** ISS to be analysed by the modelling framework, producing both text reports and visualisation of energy consumption across the network of processors in the system. The accuracy of this work is established through a series of multi-threaded, multi-core embedded software benchmarks. These are used to evaluate the effectiveness of the modelling and detail how it

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 318337, ENTRa - Whole-Systems Energy Transparency.

can be used to aid a developer’s design and implementation decisions.

Results show that the core model average error is 2.67 % with a standard deviation of 4.40 %, improving upon the prior work. The network capable model demonstrate an average error of −4.92 %, with a standard deviation of 3.92 %, supported by the VFS model with a mean squared error of 2.60 % and total error range of 15.72 %. The network model is shown to be suitable for determining the best approach for implementing two concurrent signal processing tasks on a target dual-core XS1-L platform.

Structure

The rest of this report is structured as follows. Related work is presented in Section 2, which looks at energy modelling of modern embedded processors, multi-core communication techniques and parallelism in embedded architectures, and summarises the particular implementations used in the XS1-L processor. The core- and network-level energy models are explained in Section 3, then the necessary ISS changes to support the model are presented in Section 4. Results from benchmarks exercising various parts of the model and simulation framework are discussed in Section 5 along with an evaluation of their performance in terms of accuracy and usability. Finally, Section 6 draws conclusions from this research and proposes future work.

2 Related work and background

Energy modelling of software is motivated by a need to reduce global ICT energy consumption as well as to enable devices such as embedded systems to provide more features and last longer on limited source of energy. Although hardware actually consumes energy, it does so at the behest of software, which can be inefficient if the software does not fit well to the target hardware, or does not allow the hardware to exploit its own energy saving features [20].

Multi-core systems have proliferated through ICT, from servers in datacenters down to smart phones, and now even deeply embedded systems. Any endeavour to provide software energy consumption metrics must therefore be multi-core aware. In the rest of this section we discuss related work in three background areas. First is multi-core processors in embedded systems, next is energy modelling of processors, with a focus on software level energy consumption, and finally we introduce the XS1-L processor, the particular micro-architecture used as a case study for this research.

2.1 Parallelism and multi-core embedded processors

There are various ways of realising parallelism in processors. In embedded systems, many methods have been used. VLIW (Very Long Instruction Word) has been used for some time, particularly in DSPs (Digital Signal Processors), where instruction packets enable software pipelining to be parallelised. Multi-core is becoming more prevalent, where it is beneficial to replicate a core several times and distribute work between cores. This has become necessary to provide performance gains as frequency increases have become harder to realise within practical power budgets [13].

High performance embedded processors, such as those found in smart phones, can feature multiple cores with different micro-architectures. ARM’s *big.LITTLE* is the seminal example of this, where programs can be scheduled onto simpler cores when low-energy operation is necessary or appropriate. The *little* cores are slower, but can be operated at a lower voltage and frequency point than their *big* counterparts, consuming significantly less energy. In *big.LITTLE*, significant effort is put into cache coherency between the cores, and migrating tasks can require flushing and copying of core-local caches in order to keep consistent state.

Smaller processors, such as ARM’s Cortex-M series, can also be used in multi-core, but the implementation is defined by the manufacturer. ARM has made recommendations on how to construct such devices, including cache and memory arbitration mechanisms [27]. Older generation ARM9 processors have been assembled in their thousands in the *SpiNNaker* system [5].

Rather than connecting processors via a cache hierarchy and memory bus, some systems implement a network of cores. Devices such as the Adapteva Epiphany [1] and EZChip TILE [4] processors feature many cores in a Network-on-Chip, with a grid topology of interconnects between them. In both of these processors there are multiple networks, each serving a unique purpose, such as I/O, cache coherency and direct inter-tile communication. The Intel Xeon Phi [12] uses a ring network and a hierarchy of processors, caches, tag directories and memory controllers to create a NoC that can also be viewed as a traditional memory hierarchy. Its use is not in embedded systems, but rather as a high performance computing accelerator.

The XS1-L processor features no cache hierarchy and can be assembled into a network of cores where channel style communication is possible both on- or off-chip. This is discussed in more detail in Section 2.3.

2.2 Energy modelling of processors

A program’s energy consumption is the integral of a device’s power dissipation during the course of execution:

$$E = \int_{t=0}^T P(t) dt, \quad (1)$$

although this is frequently represented using an average power, giving $E = P \times T$. To energy model a processor, P must be estimated over the course of T with sufficient granularity and precision to provide a desired accuracy. At the hardware level, detailed transistor or CMOS device models can be used, and every change in circuit state simulated to determine a fine-grained power estimation. This is time consuming and requires access to the RTL description of a processor, making this form of analysis infeasible for software developers.

Higher level models can be used instead, such as those modelling the processor as functional blocks. Instructions issued by the processor trigger activity in the functional blocks, and a cost is associated with that, which can be used to estimate the energy consumption of a sequence of instructions. At this level, the instructions are an essential part, as these drive the modelling, but also form a connection to the software — the instruction sequences for a given architecture are related to the software developer’s

program via transformation by a compiler. The ISA therefore provides a good level at which to perform analysis of hardware energy consumption at the behest of software.

Seminal work in ISA level energy modelling includes that of Tiwari et al. [26], where sequences of instructions are assigned costs, as well as the transitions between instructions, which causes circuit switching as new control paths are enabled. This work has been drawn upon to enable energy consumption simulation frameworks such as Wattch [3] and SimAnalyzer [24]. This style of ISA level modelling has also been refined to include finer grained detail on the activity along the processor data path, where data value changes also influence energy consumption [25].

Energy modelling has been performed for a wide variety of processors with various micro-architectural characteristics, for example VLIW DSP devices [10], both simple and high performance ARM variants, as well as very large processors such as modern server grade x86 devices [8] and the 61 core Xeon Phi [23]. These all draw from similar background, but account for different processor features, and obtain their model data from different sources. For example, high performance ARM and x86 models can use hardware performance counters to model activities such as cache misses, which have a significant impact on energy consumption. Simpler devices may not be so affected, and thus direct instruction level costs can be attributed. Parametrised energy models that consider properties such as operating frequency and voltage have been created for other processors, such as the Intel Xeon in [2], to inform a model predictive controller in order to smooth thermal hotspots in such dense multi core devices.

A single core model of the XMOS XS1-L architecture is presented in [15], which uses data from a series of instruction energy profiling tests in order to build the model. The architecture’s hardware multi-threading is accounted for, with the level of parallelism (active threads) contributing to energy consumption during the course of the analysed program. This model has been applied using instruction set simulation, and also via static analysis at the ISA level [7, 16] as well as the LLVM IR level [6].

2.3 The XS1-L processor and network

The XS1-L family is a group of processors implementing the XMOS XS1 ISA in a 65 nm process technology, featuring a configurable network upon which arbitrary topologies of interconnected processors can be built. Each core has a four stage pipeline and support for up to eight hardware scheduled threads. A thread can have no more than one instruction in the pipeline at any given time, therefore the XS1-L parallelism is only fully utilised if four or more threads are active.

These processors include 64 KiB of single cycle SRAM and have no cache, therefore the memory subsystem is flat and requires no special considerations with respect to timing. The majority of instructions complete in four clock cycles, with the exception of the divide and remainder instructions and any instructions that block on some form of I/O. If more than four threads are active, then the instruction issue rate per thread will reduce proportionally, but the instruction throughput of the processor remains the same. This makes timing analysis of the processor very

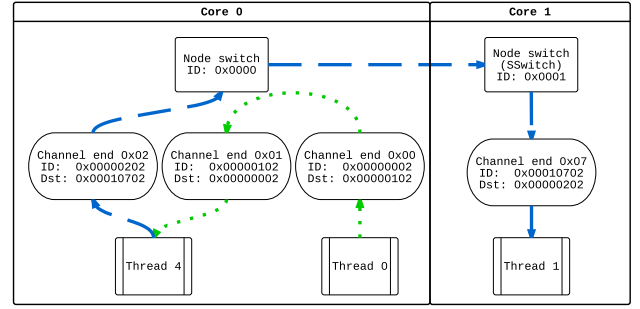


Figure 1: Visualisation of channel based communication between threads both locally and between cores.

predictable, allowing tight bounds or even exact values to be produced.

The XS1 instruction set includes provisions for *resource operations*. These are interactions with peripheral devices, such as I/O ports, synchronisers and communication channel endpoints (chanends). As such, activities such as I/O are a first class member of the instruction set. Other instruction sets, such as x86, have similar provisions [11, pp.115,176]. However, the XS1 architecture takes this further, and places these peripherals outside of the memory space, such that I/O and other resource operations are not translated into memory mapped reads and writes, but are instead a completely separate data path. This separation of memory and resources aids in the modelling processor, particularly when communication between threads and cores is considered.

On a single core, it is possible for threads to communicate or access common data using shared memory paradigms. This can be expressed in software through appropriate use of regular pointers in C, or through specially attributed pointers in version 2 of the XC language that was developed to complement XS1. However, CSP style channel communication is more prevalent in XC. Channels in XC translate into channel endpoints in the XS1 architecture, where two chanends are logically connected together. Communication then takes the form of *in* and *out* instructions. Control tokens can be used to provide synchronisation, and instructions will block if buffers are full or no data is available to read. This paradigm extends beyond core-local communication and out onto a network of cores. Therefore, concurrent programs can grow to use multiple processors with relative ease.

A network of XS1-L processors consists of multiple cores each connected to their own integrated switch. This switch provides a number of links, which can be connected to other switches, either on- or off-chip. These links can operate in either five- or two-wire mode in each direction, where the former can carry two bits per symbol and the latter one bit per symbol in an 8b/10b encoding. The five wire mode is therefore faster at the same frequency, but requires ten wires total per link. Each link possesses a receive buffer and credit based flow control is used to prevent overrun. When a link is first enabled, the sending switch must solicit credit from the switch at the other end of the link with a *hello token*. During normal operation, *credit* tokens are

sent from the receiver to the sender as buffer space becomes available.

Routing between switches is configurable based on IDs assigned to each node, where a node is a switch and its associated core. When a message is first sent from the channel of a core, the ID of the destination node is prepended to the message. Receiving switches then compare this ID to their own. If they are different, the first bit that is different is used to determine the *direction* along which the message will be routed. A direction can be assigned one or more links, and the next available link in that direction will then be used for forwarding. Typically, dimension-order routing is used to create a deadlock avoiding network, but this is dependent upon the topology network that is physically assembled. Links are held exclusively by the source channel either indefinitely, or until a closing control token is transmitted from the source. Through this approach, both wormhole routed packets and permanently reserved streaming routes can be created.

A high level view of threads communicating through channels and switches, both locally and between cores is shown in Figure 1. The precise implementation details and configuration parameters are detailed in [18, 19]. Examples of multi-core XS1 implementations include the XMP-64, which features 64 cores, using the older XS1-G family, and the Swallow project, which assembles multiple dual-core XS1-L family processors into a system of hundreds of cores [9]. These use hypercube and lattice network topologies, respectively.

3 XS1-L core and network energy model

In this report, the modelling effort of [15] is extended in several ways. Firstly, more instructions are directly energy profiled, and for those that cannot, a regression tree approach is implemented to estimate their energy cost. Secondly, additional voltage and frequency profiling is performed, using a suitable variant of the XS1-L, to produce a VFS aware model version, retaining good error bounds. Finally, network communication costs are considered, through further profiling, and a network level, communication aware model produced, integrating core, switch and interconnect components within the model.

3.1 Regression tree

The prior work of [15] investigated grouping instructions by operand count in order to provide an energy estimate for un-profiled instructions, as well as to reduce model complexity. However, the evaluation showed that this was not suitably fine grained or sufficiently accurate. Instead, each profiled instruction is accounted for individually, and un-profiled instructions are assigned a *default* value, based on the observed average of all profiled instructions.

Here, a different approach is used, where a set of instruction features are used to classify each instruction. This is combined with the direct instruction profiling data into a regression tree, allowing un-profiled instructions to receive an energy estimate based on profiled instructions with similar features.

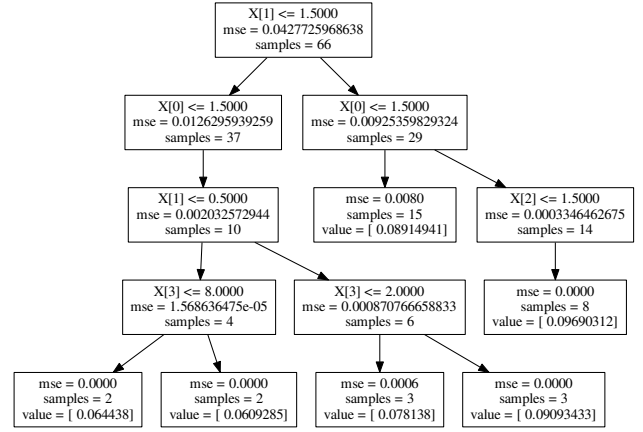


Figure 2: Visualisation of part of the model regression tree. Leaves provide energy estimations, all other nodes are decisions based on a particular instruction feature $X[f]$. Not all branches are shown; the full tree is 29 nodes.

Instr.	Features						Energy
	L	S	D	I	M	R	
add_3r	1	2	1	0	0	0	185 mW
ldc_lru6	2	0	1	10	0	0	160 mW
outct_rus	1	1	0	4	0	1	134 mW

Table 1: Input data for regression tree constructor

3.1.1 Tree construction

First, a set of features are identified, which from empirical data, demonstrate a correlation with energy consumption. These are specific to the XS1-L processor, although can be re-defined for other processors in order to re-use the technique. In the case of the XS1-L, the features are:

L: Instruction length (short or long: 1 or 2).

S: Number of source registers (count: 0–4).

D: Number of destination registers (count: 0–2).

I: Length of immediate operand (num. bits: 4–16).

M: A memory operation is performed (Boolean).

R: A resource operation is performed (Boolean).

The *Scikit-Learn DecisionTreeRegressor* [21] is used to build the regression tree. The data is presented as an matrix of instruction features and a vector of measured energy for each profiled instruction. From this, a regression based decision tree is constructed. A sample of the input data is provided in Table 1.

The regression tree construction library uses floating point feature parameters. For the given feature set, both the integer and Boolean features can be converted to their nearest floating point equivalent without consequence.

3.1.2 Tree traversal

A cutting of the regression tree for our energy model is depicted in Figure 2. When the energy cost of an instruction must be determined, a check first determines if a direct energy measurement exists. If so, it can be used within the model equation. If not, then the instruction’s features are used to traverse the decision tree. Each feature is indexed numerically by the `DecisionTreeRegressor`, and map to the features in the order we have declared them.

For example, the first decision, at the root of tree, is dependent upon the number of source operands. Those with fewer than two (or ≤ 1.5) follow the left branch, whilst those with two or more follow the right branch. The instruction length is considered next. However, descending into the tree further, the feature selection that minimises the mean squared error (mse), will differ depending on the instruction and the collected energy data. This makes the decision tree more versatile than a flat ordinary least squares regression. For example, there are no instructions with four source operands that use memory, therefore $X[4]$ or feature **M** has no influence upon such instructions. The tree is also unbalanced; some branches reach leaves in fewer levels, due to no variation in features beyond a certain decision point.

The accuracy of this approach is tested and evaluated in Section 5, where a reduction in both error and variance is shown when compared to the previous model.

3.2 VFS modelling

The XS1-L series of processors can dynamically adjust their core clock frequency when idle, and some devices support variable voltage. The low speed of voltage adjustment makes dynamic voltage and frequency scaling (DVFS) impractical for most of the real-time embedded tasks targeted to the XS1-L. However, it is still possible to statically select a best voltage and frequency for a given set of tasks, and so there is motivation to provide an energy model can support this exploration in order to determine what savings can be made.

An XMOS *SLICEKIT-A16* is used for VFS profiling, a board containing an XS1-A16 processor. The A16 contains two XS1-L family processor cores, as well as an analogue component block containing components such as ADCs and most importantly configurable DC-DC power supplies, one of which services the cores. The *SLICEKIT-A16* board provides two shunt resistors for power sensing, one for the 3.3 V I/O used by the chip and one for the 3.3 V supply fed to the on-chip voltage regulators. Measurements are performed with a MAGEEC power measurement board [17], which provides 2MSPS at 12-bit resolution with a noise floor at approximately 0.1 % of measured power in our test setup, depending on the current supplied. The measurement setup and power supply structure is different to that used in [15], so the power supplies must be considered in the new model.

3.2.1 Profiling method

VFS profiling is performed by a series of tests both at idle and high power, where each test is performed at different voltage and frequency operating points. Three configurable parameters are exercised:

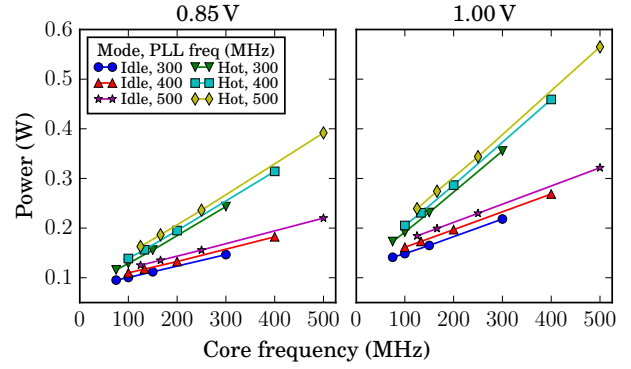


Figure 3: Power measurements at two voltage points for idle and high power tests over a range of system frequencies and dividers.

System frequency The frequency produced by the PLL, which is integrated into each processor’s switch. This sets the frequency of the switch and core, each of which can then be divided further.

Core divider The divider applied to the system frequency to produce the core frequency. Typically this is zero. The specified divider can be applied dynamically when there are no active threads, or permanently.

Core voltage The power supply to the device’s cores is configurable in 10 mV steps.

Other parameters, such as the reference clock, can be also be changed. However, the reference clock is used for timing ports and other synchronisation activities, thus we keep it at its default of 100 MHz to prevent unexpected timing changes in programs. As a result of this, profiling is limited to system frequencies of 500, 400 and 300 MHz and a core divider in the range 0–3. Error free operation was achieved with a core voltage range of 0.85–1.15 V. However, the vendor only certifies devices for operation at 1.0 V, and extensive testing at each voltage point was not performed; they were used purely for VFS characterisation.

3.2.2 VFS profiling data

Figure 3 shows the profiling data for two of the voltage points that were tested. Each plot shows six series; three for idle tests and three for high power tests, each at one of three system frequencies. Points along the x axis determine the core frequency after the divider is applied.

The 100 MHz operating point is achieved twice during tests, with $F = \frac{400}{4}$ and $F = \frac{300}{3}$. From this we see that there is an overhead in having a higher system clock, regardless of the resultant core clock. This is intuitive, as there is still some part of the system operating at the higher frequency.

3.2.3 Energy model

To produce a VFS capable energy model, we incorporate the configurable parameters defined in Section 3.2.1 into a suitably modified model equation. Curve fitting is used to

determine the contribution that these parameters have to the equation.

SciPy's Nelder-Mead method [22] is used to minimize the error of the function Equation (2) against the idle test profile data collected, for the following parameters. C_{pll} is the characteristic capacitance present at the system frequency (or PLL frequency). C_{idle} is the characteristic capacitance in the core at idle. I_{leak} is the static leakage current. Finally, I_{ext} captures other effects parametric to the supply voltage and scaled by the power dissipated in the device, approximating power supply efficiency.

$$F = (V^2 C_{\text{pll}} F_{\text{pll}} + V^2 C_{\text{idle}} F_{\text{core}} + V I_{\text{leak}}) \times V I_{\text{ext}} \quad (2)$$

The resultant parameters are:

$$\begin{aligned} C_{\text{pll}} &= 675 \times 10^{-12}, & C_{\text{idle}} &= 1.68 \times 10^{-9} \\ I_{\text{leak}} &= 334 \times 10^{-3}, & I_{\text{ext}} &= 106 \times 10^{-3}. \end{aligned} \quad (3)$$

A parameter is also determined for the high-power tests, $C_{\text{hot}} = 2.15 \times 10^{-9}$, although it is used only for validation, and not in the final energy model. Testing these parameters against the profiling data, a mean squared error of 2.60 % is achieved. The minimum error is -3.58 % and the maximum 12.14 %, giving a full error range of 15.72 %.

These parameters are then used in a modified version of the instruction level energy model from [15]. This yields Equation (4) as the new model:

$$\begin{aligned} E_{\text{instr}} &= (V^2 C_{\text{pll}} F_{\text{pll}} \\ &\quad + V^2 F_{\text{core}} (C_{\text{idle}} + C_{\text{instr}}) M_{N_{\text{pipe}}} O \\ &\quad + V I_{\text{leak}}) \times V I_{\text{ext}} \times 4 T_{\text{clk}} \end{aligned} \quad (4)$$

where $M_{\text{pipe}} = \min(4, N_t)$.

This captures the previous components of the model, plus the frequency and voltage dependent parameters. N_{pipe} is the number of threads present in the pipeline when the instruction's energy is measured, which is the minimum of 4 and the number of active threads. This is used select a scaling factor due to parallelism, M . An average inter-instruction overhead, O , is also included, as per the original model.

The remainder of this report focuses more on network aware energy modelling, rather than VFS design space exploration. Future work could include exercising the VFS aspect of the energy model more heavily, and so is included here for the benefit of such endeavours.

3.3 Network modelling

A system level model of a network of XS1-L processors is comprised of multiple core model instances, as well additional modelling components to capture network switch and link activity. The core model requires either simulated instruction sequences or appropriately parametrised static analysis. A system level model must support the interaction between multiple cores. The implementation details of this at a simulation level, are covered in Section 4.

3.3.1 Parameters

Communication costs must be accounted for in three system components. Firstly, the core, where the *in* and *out* instructions are executed. These are already captured in

the core energy model. Second is the switch, which consumes energy as it routes tokens through it. Finally, the interconnects over which tokens are transmitted must be considered.

Switch energy consumption data is acquired from profiling of the larger Swallow XS1-L system [14, p. 124]. Link energy for the SLICEKIT-A16 is determined from direct profiling. These are shown in terms of Joules per token in Equation (5).

$$E_{\text{switch}} = 70.8 \times 10^{-12} \text{ J}, \quad E_{\text{link}} = 221 \times 10^{-12} \text{ J} \quad (5)$$

Currently, these are fixed values. However, it is possible to parametrise these by link length (where longer wiring has a higher capacitance), as well as by switch frequency and voltage. This may form future work, joining well with the proposed further work on VFS modelling.

3.3.2 Construction

The network level model is constructed using the **NetworkX** library for Python, which allows networks with nodes and edges that have arbitrary attributes. The XML file used by the developer or vendor to describe an XMOS based system (the XN file), is read by the energy modelling framework and used to construct a graph of the system's cores, switches and links.

When a simulation trace is analysed by the modelling tool, energy is incremented in each graph node or edge as appropriate. Instructions increase the core energy of the relevant core, whilst token traces increase the source switch and traversed link energy. For this trace analysis to account for network activity, the trace must include network activity that identifies tokens traversing links. This change was made to *axe* as part of the modifications described in Section 4.

At the end of the modelling run, this data can be aggregated into a text report, broken down by core, or as a visualisation. These will be shown in Section 5.

4 ISS network and timing implementation

XS1-L energy modelling has been demonstrated using statistics from instruction set simulation as well as various levels of static analysis. Using full instruction set simulation traces provide more detail, at the cost of simulation time. However, by improving analysis of traces to complete once a *function or section of interest* has completed, simulation time can be kept low. The same triggering methods used by the hardware measurement, can easily be used to define sections of interest by identifying the relevant I/O resource instructions in a trace. This means single iterations of functions or algorithms can be observed by modelling, where repeated iterations are required for physical measurement. The slowdown of simulation is mitigated to some degree by this. This is mitigated further by the use of *axe*, an open source XS1 simulator that is faster than its closed source *xsim* counterpart, although it can be less accurate. A number of *axe* enhancements are detailed in this section that improve its accuracy, whilst preserving some of its performance advantage.

Enabling full trace simulation allows better debugging of the energy model, as well as the opportunity to more closely scrutinise where energy is being consumed. To that end, this work focuses on full traces. However, the models underpinning this work can be adapted for use at other levels of abstraction, as with previous model versions.

4.1 Instruction scheduling

The modified version of **axe** enforces strict instruction scheduling, where each active thread may only issue one instruction before the next queued thread is given an opportunity. The timestamps of subsequent instructions in a thread are incremented by $\min(4, N_t)$, to reflect the four-stage, hazard free pipeline.

This more closely follows the micro-architecture, whereas the original **axe** implementation may issue multiple instructions from one active thread even when another thread is also in an active state. This also ensures that the timestamps in instruction traces are ordered, greatly simplifying the process of determining pipeline occupation during energy modelling.

4.2 FNOP simulation

In addition to instruction scheduling changes, occurrences of fetch no-ops (FNOPs) are also recorded in the modified simulator. A simple model of the processor’s instruction buffers is used to determine when a thread must stall in order to fetch the next instruction word. The conditions leading to an FNOP include:

- Sequences of memory operations in a thread, preventing any instructions being fetched for that thread during the memory stage of the pipeline.
- Branching to an unaligned 4-byte instruction, where only the first half of the instruction is fetched during the memory stage of the branch instruction.

Many FNOPs can be avoided by re-scheduling long instructions and memory instructions amongst short, non-memory instructions, as well as word aligning entry points to loops where the first instruction is long. However, the compiler does not currently do all of these automatically. The impact of FNOPs can be significant if they are present in tight loops, increasing execution time and therefore energy. Thus, is it important to correctly simulate this behaviour for accurate energy modelling.

4.3 Switch and link control flow

Both the **axe** and **xsim** XS1-L instruction set simulators support channel communication in multi-core programs. However, even with accurate core-local instruction scheduling, neither include accurate simulation of network behaviour. At a functional level, link utilisation and route reservation are simulated, such that protocol violations can be detected and appropriate exceptions raised, as well as some degree of performance limiting due to route contention. However, multi-core message tokens are transmitted in zero time. This can create significant timing error when simulating communicating multi-core programs.

To address this, we have added a model of the link control flow mechanisms from XS1-L into **axe**’s switch, link

Test	Recorded time (μ s)			Error (%)	
	HW	xsim	axe	xsim	axe
One core	2.270	1.118	2.272	-50.75	0.09
Two core	8.300	2.150	8.287	-74.10	-0.16

Table 2: 1024-word message timing, comparing dual-core hardware to **xsim** and modified **axe** simulators.

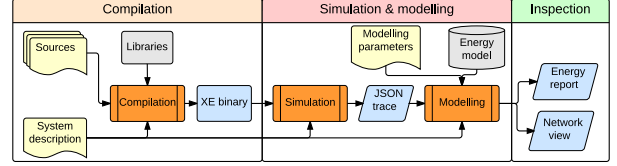


Figure 4: Energy aware multi-core software development workflow.

and channel end code. Control tokens such as HELLO and the initial CREDIT issue [19, pp. 10–13] are now handled rather than ignored. Symbol and token delays on links, as specified in the XN platform description file at compile time, are also obeyed. This ensures that messages traverse the network in a realistic time-scale, and that as buffers fill, network throughput is throttled.

The current changes are not completely faithful to the hardware, but yield a significant improvement on the previous simulation capabilities. This is evident in Table 2, which compares a 1024-word transfer between two channel ends on a SLICEKIT-A16 in both core-local and dual-core variants, with respect to the actual hardware timing, **xsim** simulation and the modified **axe** simulation. The error in **xsim** exceeds 50 % in both cases, whereas **axe** achieves less than 0.2 %. Over a broader range of similar tests, with different message lengths and producer/consume rates, **axe** is able to maintain an average error of 0.80 % with a standard deviation of 1.26 %.

To aid modelling, switch and link activity are added to **axe** simulation traces. These resemble the switch tracing present in the vendor’s **xsim** simulator with the `--tracing-switch` parameter set, but are in a JSON format that is more readily consumable by the energy modelling framework.

5 Benchmarking and evaluation

This evaluation considers enhancements to the core model as well as the multi-core communication model. However, VFS modelling, as discussed in Section 3.2, remains for future work, due to the current instruction set simulation framework not supporting configurable operating frequencies and clock dividers without significant further development. This does not exclude the VFS model from use in other forms of non-simulation based analysis, however.

5.1 Core benchmarks

The extended core energy model is evaluated using the same benchmark suite as the original model in [15] and on the same single core XS1-L hardware. This provides direct

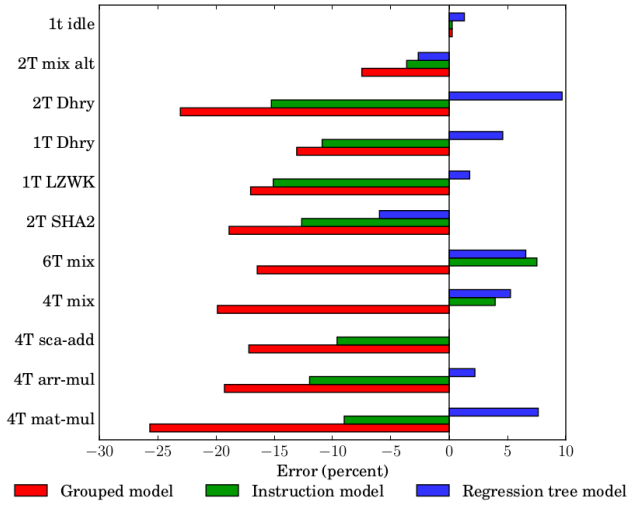


Figure 5: Error of previous (grouped, instruction) models versus new (regression tree) model.

Model version	Error (%)	Std. dev. (%)
Grouped	-16.42	6.91
Instruction	-7.23	7.45
Regression tree	2.67	4.40

Table 3: Geometric mean model error and standard deviation of the tested energy models.

comparison between the accuracy of both model versions with respect to the target hardware. The benchmarks used include the system at idle, various audio sample mixing variants, operating on multiple independent streams with various levels of concurrency, one and two concurrent Dhrystone instances, as well as multi-threaded parallel matrix operations. They are explained in more detail in [15, pp. 20–21].

Figure 5 shows the model error for each benchmark with respect to the hardware measured energy consumption. The regression tree model performs better than both of the previous model versions in the majority of benchmarks. Where the original instruction model out-performs the regression tree model, the difference is approximately a single percentage point of error. The average and standard deviation of the errors is summarised in Table 3, where it is evident that the regression tree model improves overall accuracy whilst reducing variance and the overall range of error across benchmarks.

5.2 Multi-core benchmarks

To test the multi-core model, suitable benchmarks must be used. Those used to test the core model do not lend themselves to multi-core deployment, due to their structure and limited, if any, use of channel communication. Instead, two new applications are used for multi-core benchmarking. These are a Finite Impulse Response filter (FIR), and the Infinite Impulse Response (IIR) Biquad filter, which will be referred to **fir** and **biq** respectively.

Both of these benchmarks are used for applying signal processing, in this case to streams of audio samples. This is

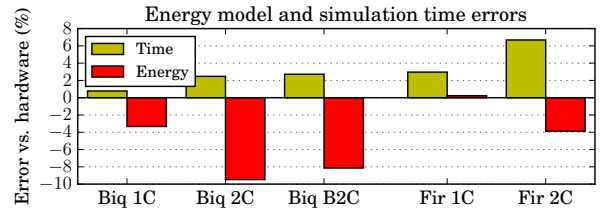


Figure 6: Time and energy errors for **fir** and **biq** benchmarks in single (1C) dual core (2C) and for **biq**, bad dual core (B2C) configurations.

Property	G. mean (%)	Std. dev. (%)
Time	3.10	2.16
Energy	-4.92	3.92

Table 4: Geometric mean and standard deviation of timing simulation and energy model errors for all **biq** and **fir** benchmarks.

a typical application area for the target processor, and so a good benchmark selection. Both benchmarks feature multiple stages, **fir** implementing seven taps and **biq** featuring seven individual biquads.

The concurrent implementation of these applications represent each stage as a thread, with the progressively filtered samples passed over channels between threads. The seven threads can be allocated onto a single core the SLICEKIT-A16, or distributed across the device’s two cores. We test both 7 : 0 and 4 : 3 thread distributions between the two cores, giving consideration to the fact that both thread and processor instruction throughput are optimal when a core has four active threads. We also implement a poorly allocated version of **biq** where threads communicate between cores sub-optimally.

In Figure 6 the energy and timing errors for the benchmarks are presented. We observe that in all cases, the simulation over-estimates execution time, but by less than 7%. the energy model under predicts in the majority of cases, but remains within a 10% error margin.

The overall results are summarised in Table 4, which demonstrates single-digit percentage mean and standard deviations for the errors. Note that the timing over-prediction counteracts the energy model under-prediction. Improving one in isolation may in fact reduce the visible accuracy of the process overall. It is therefore essential to examine the multiple dimensions of error that are present, in order to direct effort appropriately.

Figure 7 shows a visualisation of energy consumption in the cores, switches and interconnect of the SLICEKIT-A16. Cores are annotated with the modelled energy consumption, and switches show their energy consumption as well as the aggregated energy consumed on outbound links. Links are also coloured by energy. The colouring of the graphs has been scaled to be directly comparable. Hot/cold are represented as pink/blue for switches and green/red for cores. Links turn orange as they consume more energy. Only links between switches are energy modelled, as the

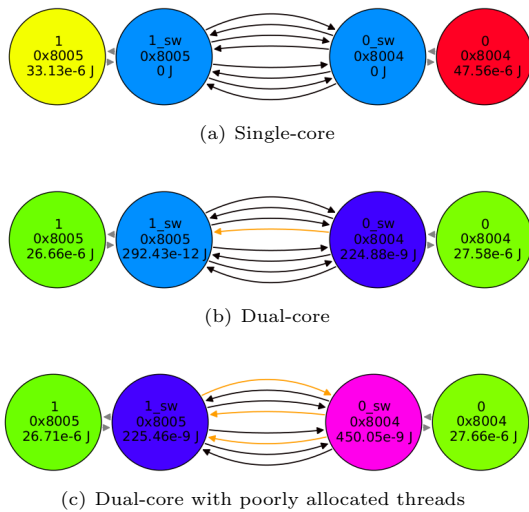


Figure 7: Network level energy consumption visualisations the **biq** benchmark.

core to switch links are captured implicitly within the core model.

Although visualisation is a less precise representation, it does allow for comparison and inspection in order to determine *where* energy is consumed. From these examples, we see that the single core implementation in fig. 7(a) is the least efficient, taking more energy on the active core, and resulting in significant energy consumption from leakage in the otherwise idle core. In fig. 7(b), the benchmark completes quicker and work is distributed, so the cores consume less total energy. The communication cost is insignificant in comparison; some three orders of magnitude less. Finally, fig. 7(c) is again dual core, but allocates the software pipeline stages poorly, resulting in three times more core-to-core communication. The cores consume slightly more energy due to a marginally longer run-time, and the network cost is three times higher. Not only does this demonstrate the desirability of distributing the workload across the available cores, it also demonstrates that energy inefficiency can be introduced with minimal timing impact, where communication latency may be hidden.

6 Conclusions and future work

In this work, a single core, multi-threaded energy model is presented with an average error of less than 5%. This is enabled through both instruction set simulator enhancements and a regression tree approach to modelling instructions that cannot be directly energy profiled.

A multi-core model is then described and tested, again supported by instruction set simulation enhancements. The timing error of the simulator is shown to be within 7% and the energy estimation error within -10% for two multi-core audio filtering benchmarks, with average errors of 3.10% and -4.92% respectively.

This combination of tools and the demonstrated workflow allows for multi-threaded, multi-core software design space exploration, in order to establish which software definable

properties, such as thread allocation and communication patterns, impact the energy consumption of a target device.

A voltage and frequency scaling adaptation of the core energy model is also presented, with a mean squared error of 2.6%. In future work, we propose that the **axe** simulation tool can be further improved to support frequency selection, allowing instruction set simulation, VFS-aware energy modelling, and thus design exploration including VFS parameters.

Additional opportunities for future work include incorporating these new models into static analysis techniques. Some static analysis techniques have been demonstrated against prior work on multi-threaded energy models [16, 7, 6], and so there is a strong case for extending such work to the multi-core level. Simulation based modelling could also be further extended by examining larger systems, such as the many-core Swallow system [9], which assembles potentially hundreds of XS1-L processors into a compute grid. However, appropriately sized benchmark applications, or compositions of smaller related tasks, would need to be identified, adapted or developed in order to exercise the model and such a system in an appropriate context.

References

- [1] Adapteva. *Ephiphany Introduction*. 2015. URL: <http://www.adapteva.com/introduction/> (visited on 03/30/2015).
- [2] A. Bartolini et al. “A Distributed and Self-Calibrating Model-Predictive Controller for Energy and Thermal management of High-Performance Multicores”. In: *Energy* (2011).
- [3] D. Brooks, V. Tiwari, and M. Martonosi. *Wattch: A Framework for Architectural-Level Power Analysis and Optimizations*. May 2000. DOI: 10.1145/342001.339657. URL: <http://portal.acm.org/citation.cfm?doid=342001.339657>.
- [4] EZChip Semiconductor. *TILE-Gx72 Processor*. Tech. rep. 2009, pp. 1–2.
- [5] S Furber et al. “Overview of the SpiNNaker System Architecture”. In: *IEEE Transactions on Computers* 62.12 (2013), pp. 2454–2467. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6226357.
- [6] K. Georgiou, S. Kerrison, and K. Eder. *A Multi-level Worst Case Energy Consumption Static Analysis for Single and Multi-threaded Embedded Programs*. Tech. rep. University of Bristol, 2014. URL: http://www.cs.bris.ac.uk/Publications/pub_master.jsp?id=2001701.
- [7] N. Grech et al. “Static analysis of energy consumption for LLVM IR programs”. In: *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*. SCOPES ’15. Sankt Goar, Germany: ACM, 2015. DOI: 10.1145/2764967.2764974.
- [8] W. Heirman et al. “Power-aware multi-core simulation for early design stage hardware/software co-optimization”. In: *Proceedings of the 21st international conference hardware/software co-optimization - PACT ’12*. New York, New York, USA: ACM Press, 2012, p. 3. ISBN: 9781450311823. DOI: 10.1145/2370816.2370820. URL: <http://dl.acm.org/citation.cfm?doid=2370816.2370820>.
- [9] S. J. Hollis and S. Kerrison. “Overview of Swallow - A Scalable 480-core System for Investigating the Performance and Energy Efficiency of Many-core Applications and Operating Systems”. In: *arXiv* (2015). URL: <http://arxiv.org/abs/1504.06357>.

- [10] M. Ibrahim, M. Rupp, and S. Habib. *Power consumption model at functional level for VLIW digital signal processors*. Tech. rep. 1. 2008, pp. 2–7. URL: http://www.researchgate.net/publication/228947933/_Power/_consumption/_model/_at/_functional/_level/_for/_VLIW/_digital/_signal/_processors/file/e0b49521c2bc72bd43.pdf.
- [11] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes*. December. 2011, p. 3463.
- [12] Intel Corporation. *Intel Xeon Phi Coprocessor*. Tech. rep. 2013.
- [13] A. B. Kahng. “The ITRS design technology and system drivers roadmap”. In: *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*. New York, New York, USA: ACM Press, 2013, p. 1. ISBN: 9781450320719. DOI: 10.1145/2463209.2488776. URL: <http://dl.acm.org/citation.cfm?doid=2463209.2488776>.
- [14] S. Kerrison. “Energy modelling of multi-threaded, multi-core software for embedded systems”. PhD thesis. University of Bristol, Dept. of Computer Science, 2015.
- [15] S. Kerrison and K. Eder. “Energy Modeling of Software for a Hardware Multithreaded Embedded Microprocessor”. In: *ACM Transactions on Embedded Computing Systems* 14.3 (Apr. 2015), 56:1–56:25. ISSN: 1539-9087. DOI: 10.1145/2700104. URL: <http://doi.acm.org/10.1145/2700104>.
- [16] U. Liqat et al. “Energy Consumption Analysis of Programs based on XMOS ISA-Level Models”. In: *23rd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'13)*. Springer, Sept. 2015.
- [17] MAGEEC Project. *MAGEEC Power Measurement Board*. 2014. URL: http://mageec.org/wiki/Power_Measurement_Board (visited on 07/04/2015).
- [18] D. May. *The XMOS XS1 Architecture*. 2009. ISBN: 9781907361012.
- [19] D. May et al. *XS1-L System Specification*. 2008.
- [20] K. Roy and M. C. Johnson. “Software design for low power”. In: *Low power design in deep submicron electronics*. Kluwer Academic Publishers, 1997. Chap. 6, pp. 433–460. ISBN: 0-7923-4569-X. URL: <http://dl.acm.org/citation.cfm?id=265902>.
- [21] Scikit-Learn. *Scikit-Learn Decision Trees*. 2015. URL: <http://scikit-learn.org/stable/modules/tree.html> (visited on 03/18/2015).
- [22] SciPy. *scipy.optimize.minimize — SciPy v0.15.1 Reference Guide*. 2015. URL: <http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.optimize.minimize.html> (visited on 08/12/2015).
- [23] Y. S. Shao and D. Brooks. “Energy characterization and instruction-level energy model of Intel’s Xeon Phi processor”. In: *International Symposium on Low Power Electronics and Design (ISLPED)*. November. IEEE, Sept. 2013, pp. 389–394. ISBN: 978-1-4799-1235-3. DOI: 10.1109/ISLPED.2013.6629328. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6629328>.
- [24] Sim-Panalyser. *Sim-Panalyser 2.0 Reference Manual*. 2004, pp. 1–54.
- [25] S. Steinke et al. “An accurate and fine grain instruction-level energy model supporting software optimizations”. In: *Proc. of PATMOS*. Citeseer, 2001. DOI: 10.1.1.115.3528. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.6971&rep=rep1&type=pdf>.
- [26] V. Tiwari, S. Malik, and A. Wolfe. “Compilation techniques for low energy: An overview”. In: *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*. IEEE, 1994, pp. 38–39. ISBN: 0780319532. URL: http://ieeexplore.ieee.org/xpls/abs/_all.jsp?arnumber=573195.
- [27] J. Yiu and I. Johnson. *Multi-core microcontroller design with Cortex-M processors and CoreSight SoC*. Tech. rep. ARM, 2013.